Learning pattern languages using queries

AUTHOR(S):

MATSUMOTO, Satoshi; SHINOHARA, Ayumi

CITATION:

MATSUMOTO, Satoshi ...[et al]. Learning pattern languages using queries. 数理解析研究所講究録 1997, 992: 58-65

ISSUE DATE:

1997-05

URL:

http://hdl.handle.net/2433/61156

数理解析研究所講究録
992巻 1997年 58-65

Learning pattern languages using queries

松本 哲志* 篠原 歩

Satoshi MATSUMOTO Ayumi SHINOHARA

Department of Informatics,
Kyushu University 33, Fukuoka 812-81
e-mails:{matumoto, ayumi}@i.kyushu-u.ac.jp

Abstract

A pattern is a finite string of constant and variable symbols. For $k \geq 1$, we denote by $k\mu\Pi$ the set of all patterns in which each variable symbol occurs at most k times. In particular, we abbreviate $\mu\Pi$ for $k = 1$. The language $L(\pi)$ of a pattern π is the set of all strings obtained by substituting any non-null constant string for each variable symbol in π. In this paper, we show that any pattern $\pi \in k\mu\Pi$ is exactly identifiable in $O(|w|^{k+2})$ time from one positive example $w \in L(\pi)$ using $|w|^{k+1} + |\pi|^k$ membership queries. Moreover, we introduce the notion of critical pattern, and show that the number of membership queries can be reduced to $|w| + |\pi|$ if the target pattern $\pi \in \mu\Pi$ is not critical. For instance, any pattern $\pi \in \mu\Pi$ whose constant parts are of length at most 3 is not critical. Finally, we show a nontrivial subclass of $\mu\Pi$ that is identified using membership queries only, without any initial positive example.

1 Introduction

We investigate the feasibility of learning *pattern languages*, in the setting of exact identification using queries [4]. A pattern is a finite string of constant and variable symbols. A pattern language, which was introduced by Angluin [1], is the set of all strings obtained by substituting any non-null constant string for each variable symbol in the pattern. Because of its intuitive clearness of the expression and some desirable properties as formal languages, the learnability of pattern languages from examples has been studied in various settings. For example, pattern language learning algorithms have been proposed within the inductive inference model [1, 11, 20], the query learning setting [4, 11, 12], the PAC-learning model [9, 14]. Additionally, the average-case behavior of pattern language learners has been investigated [25], and applications in Genome Informatics have been proposed [5, 7, 21, 22].

Learning a concept from *good* examples has attracted considerable attention in recent studies [10, 11, 15, 17, 24], since it provides the best case analysis as well as the limits of learnability. Possibly, the *goodness* of the examples given heavily dependents on the properties of the languages to be identified. Informally, an example is regarded to be *good* if it provides a *significant* information for learning the concept.

Concerning the pattern languages, a positive example gives a learner crucial information to identify the target pattern, since the hypothesis space can be immediately restricted to a finite set [1, 2]. This is a key property for showing the class of pattern languages to be identifiable from positive examples in the framework of inductive inference [1, 2, 20]. Without any positive example, it is hard to learn a target pattern efficiently. Angluin [1] showed that the class of pattern languages is not identifiable in polynomial time using both membership and equivalence queries. The trick behind the adversary

* This author is a Research Fellow of the Japan Society for the Promotion of Science (JSPS). The author's research is partly supported by Grants-in-Aid for JSPS research fellows from the Ministry of Education, Science and Culture, Japan.

teacher is to answer equivalence queries without returning any positive example to the learner. These results provide evidence that good examples should be positive ones.

Marron [12] refined this line of research by studying pattern language learner that initially receives one positive example. Any other information concerning the target must be obtained by asking membership queries. Thus, the goodness of the initial example can be measured by the number of membership queries additionally needed until successful learning. The result suggests that *all but small fraction* of positive examples are *good* to learn a k-variable pattern.

Our goal is a bit more ambitious in that we would like to measure the information content provided by a positive example in dependence on its length. Intuitively, *the shorter the given positive example is, the more informative to the learner it is*, since the hypothesis space can be restricted to be a small set of patterns if the positive example is short. Especially, if the positive example is known to be a *shortest* one, the learner has only to find the positions where variable symbols occur in the target pattern. Based on the above observation, we propose a simple learning algorithm which consists of the following two phases: at the first phase, keep trying to *shrink* the given positive example by asking membership queries, until confirmation that the resulting example is a shortest one. In the second phase, identify the positions at which variable symbols occur in the target pattern. The second phase is a relatively easy task for the classes we will deal with, as we shall briefly state in Section 3. Therefore, the main part of this paper is devoted to the first phase: *how to get a shortest positive example efficiently using membership queries*, from any given (possibly very long) positive example. The process is regarded as refining the hypothesis space by shrinking the given positive example.

While our algorithm works in general, it is not necessarily efficient (measured by the number of membership queries needed). Thus, we introduce a syntactical measure, namely the maximum number of occurrences of each variable symbol in the pattern, in order to isolate the polynomially learnable subclass of pattern languages. Let $k\mu\Pi$ be the set of all patterns where each variable symbol occurs at most k times, and let $k\mu\mathcal{PAT}$ be the corresponding class of pattern languages. In particular, we abbreviate $\mu\Pi$ and $\mu\mathcal{PAT}$ for $k = 1$. In Section 3, we show that any pattern $\pi_* \in k\mu\Pi$ can be identified from *any* initial positive example $w \in L(\pi_*)$ using at most $|w|^{k+1} + |\pi_*|^k$ membership queries. Provided each membership query is answered in unit time and assuming the usual RAM-model [1], the resulting learning time is $O(|w|^{k+2})$. This result nicely contrasts the fact that without membership queries, $k\mu\mathcal{PAT}$ is not polynomial-time PAC-learnable even for the case $k = 1$ unless RP=NP [14]. Note that *without any initial positive example*, all the classes $k\mu\mathcal{PAT}$, $k \geq 1$, are not polynomial-time query-identifiable using both membership and equivalence queries, since we can construct a malicious teacher as in [4].

Moreover, we pay special attention to subclasses of $\mu\mathcal{PAT}$: First, reduction of the number of membership queries. Second, we ask whether the initial positive example can be dropped. A pattern in $\mu\Pi$ is called *regular*, since the generated language is regular [18, 19]. The class $\mu\Pi$ is useful for practical applications [5, 14, 16], since membership queries can be answered in linear time, while for general patterns the membership problem is NP-complete [1].

In Section 4, we reduce the number of membership queries by restricting the length of the constant parts in the pattern $\pi \in \mu\Pi$. We show that any pattern in $\mu\Pi$ whose constant parts are of length at most 3 can be exactly identified in time $O(|w|^2)$ using exactly $|w| + |\pi_*|$ membership queries. This is a natural extension of learning *subsequence languages* using membership queries as investigated in [13].

Section 5 shows a nontrivial subclass of patterns in $\mu\Pi$ with fixed length of constant parts, where a learner can enumerate effectively the candidates for positive examples: That means, any pattern in this class can be identified using membership queries only *without any initial positive example.* This is also a natural extension of the good property for subsequence languages [13].

2 Preliminaries

Let Σ be a finite alphabet, i.e., a nonempty finite set of *constant symbols*, and $X = \{x, y, \ldots\}$ be a set of *variables symbols*. We assume that $\Sigma \cap X = \phi$ and Σ contains at least two constant symbols. For an alphabet Δ, let Δ^+ denote the set of all nonempty strings and $\Delta^{[n]}$ the set of all strings of the length at most n for $n \geq 0$. We denote by $w[i]$ the i-th symbol in a string w, and by $w[\tilde{i_1}, \cdots, \tilde{i_l}]$ the string which is obtained by eliminating $w[i_1], \cdots, w[i_l]$ from w. By $w[i:j]$, we denote the substring

$w[i]\cdots w[j]$ of w. For convenience, a prefix $w[1:i]$ is abbreviated as $w[:i]$, and a suffix $w[i:|w|]$ as $w[i:]$.

A *pattern* is a string in $(\Sigma \cup X)^+$. The set of all patterns is denoted by Π. For a pattern π, the length of π is the number of symbols composing it, and is denoted by $|\pi|$. We denote by $k\mu\Pi$ the set of all patterns in which each variable symbol occurs at most k times. Note that the class $k\mu\Pi$ is different from the set of *k-variable patterns*, in which at most k kinds of variable symbols appear.

A *substitution* θ is a homomorphism from patterns to patterns such that $\theta(a) = a$ for each $a \in \Sigma$ and each variable symbol is replaced with any pattern. For a pattern π and a substitution θ, we denote by $\theta(\pi)$ the image of π by θ. The *pattern language of* π, denoted by $L(\pi)$, is the set $\{w \in \Sigma^+ \mid w = \theta(\pi) \text{ for some substitution } \theta\}$. We denote by $\mathcal{PAT}$ the class of all pattern languages, and by $k\mu\mathcal{PAT}$ the class of all pattern languages $L(\pi)$ such that $\pi \in k\mu\mathcal{PAT}$. In particular, we write $\mu\Pi$ and $\mu\mathcal{PAT}$ for $k = 1$. A pattern π in $\mu\Pi$ has been called a *regular pattern* in the literature [14, 19], because the language $L(\pi)$ is regular.

In the sequel, let π_* be a *target pattern* to be identified. A string w is said to be a *positive example* if $w \in L(\pi_*)$ and a *negative example* if $w \notin L(\pi_*)$.

The problem to find a consistent pattern $\pi \in \mu\Pi$ from given positive and negative examples is shown to be NP-complete [14]. Thus the class $\mu\mathcal{PAT}$ is not polynomial-time learnable in the PAC-learning model [23], under the assumption of NP $\neq$ RP.

To overcome this computational hardness, we allow learning algorithms to use *membership queries* [3, 4, 6, 8]. A membership query is to propose a string w. The answer is "*yes*" if $w \in L(\pi_*)$, and "*no*" otherwise. We denote by $Mem_{L(\pi_*)}$ the *membership oracle* which answers membership queries about the language $L(\pi_*)$. We assume that it takes $|w|$ steps to use a membership query for $w \in \Sigma^*$. It is known that the class $\mathcal{PAT}$ is *not* learnable using *membership queries only* [4]. We can show that for any $k \geq 1$, the class $k\mu\mathcal{PAT}$ is *not* learnable using *membership queries only* in the same way.

A *learning algorithm* $\mathcal{A}$ may collect information about π_* using one positive example $w \in L(\pi_*)$ and asking to the membership oracle $Mem_{L(\pi_*)}$. The goal of $\mathcal{A}$ is *exact identification* in polynomial time, that is, $\mathcal{A}$ must halt and output $\pi \in k\mu\Pi$ such that $L(\pi) = L(\pi_*)$, within the time polynomial with respect to $|\pi_*|$ and $|w|$.

3 Learning a pattern in $k\mu\Pi$ from one positive example using membership queries

When trying to learn an unknown target pattern π_* from examples, a positive example $w \in L(\pi_*)$ provides crucial information to a learner, since the hypothesis space can be restricted to a finite set $\{L(\pi) \subseteq \Sigma^+ \mid \theta(\pi) = w \text{ for some } \theta\}$. This is a key property to show that the class of pattern languages is identifiable from positive examples in the framework of inductive inference [1, 20]. The shorter the positive example w is, the smaller the hypothesis space is. Especially, if the positive example w is a shortest one, the learner has only to find the positions where variable symbols occur in the target pattern π_*, since $|\pi_*| = |w|$. For example, if the target pattern π_* is in $\mu\Pi$ and the given positive example w is known to be a shortest one, the learner can exactly identify π_* using $|w|$ membership queries: This is because the learner can decide whether the i-th symbol of the target pattern π_* is a variable symbol or the constant symbol by asking whether $w[!i] \in L(\pi_*)$ or not, where $w[!i]$ is a string obtained from w by replacing $w[i]$ with another constant symbol. In the same way, if the target pattern π_* is in $k\mu\Pi$ and $w \in L(\pi_*)$ is a shortest positive example, $|w|^k$ membership queries are enough to identify π_*. Therefore, in the sequel, we concentrate our attention on how to get a shortest positive example by using membership queries. We denote by $\mathcal{SPS}(L(\pi))$ the set of all shortest strings in $L(\pi)$, i.e., $\mathcal{SPS}(L(\pi)) = \{w \in L(\pi) \mid \forall v \in L(\pi), |v| \geq |w|\}$. An element in $\mathcal{SPS}(L(\pi))$ is called a *shortest positive string of* $L(\pi)$. Remark that $\mathcal{SPS}(L(\pi)) = L(\pi) \cap \Sigma^{[|\pi|]}$ for any $\pi \in \Pi$.

First we consider how to find a shortest positive string of $L(\pi_*)$ when one positive example $w \in L(\pi_*)$ is given. The basic idea is quite simple: Just ask to the membership oracle whether a subsequence w' of w is in $L(\pi_*)$ or not. If the reply is "yes", we have the shorter positive example w'. Repeat this procedure until we have confirmed that w' is a shortest one. Fig. 1 illustrates this scheme for the case $\pi_* \in \mu\Pi$.

The following lemma supports the correctness of the scheme in Fig. 1.

```
while ( w[~i] ∈ L(π*) for some i ) do
    /* Since w[i] is redundant, delete it */
    w := w[~i];
return w
```

Figure 1: The basic idea to find a shortest positive string of $L(\pi_*)$ from one positive example $w \in L(\pi_*)$ for $\pi_* \in \mu\Pi$.

Lemma 1. For a pattern $\pi_* \in \mu\Pi$ and $w \in L(\pi_*)$, $w \in \mathcal{SPS}(L(\pi_*))$ if and only if $w[\tilde{i}] \notin L(\pi_*)$ for any i.

We now analyze the complexity of the procedure which implements the scheme in Fig. 1. At the first glance, the procedure *Shrink* in Fig. 2, which checks whether each $w[\tilde{i}]$ is redundant or not *only once*, may work correctly. The running time of *Shrink* is obviously $O(|w|^2)$ since it uses exactly $|w|$

```
Procedure Shrink(w) /* shrink a positive example from left to right */
Input: a string w in L(π*).
Given: Mem_{L(π*)} for π* ∈ μΠ .
i := 1;
while (i ≤ |w|) do
  begin
    while (Mem_{L(π*)}(w[~i]) = "yes") do
      w := w[~i];
    i := i + 1;
  end
return w;
```

Figure 2: Procedure *Shrink*

membership queries.

However, unfortunately, the procedure *Shrink* does not always output a shortest positive string of $L(\pi_*)$. We present an instance where *Shrink* fails to find a shortest example.

Example 1. For a pattern $\pi_* = x\mathtt{abac}y \in \mu\Pi$ and a positive example $w = \mathtt{aabacbaca} \in L(\pi_*)$, the procedure *Shrink*$(w)$ returns the string `aababaca`, which is not a shortest.

In Section 4, we will give a sufficient condition which guarantees that the output of *Shrink* is always a shortest one. In general, for $\mu\Pi$, we need the procedure *FindShortest* in Fig. 3 which calls *Shrink* as a subroutine.

Theorem 1. For a pattern $\pi_* \in \mu\Pi$ and a string $w \in L(\pi_*)$, the procedure *FindShortest* returns a string $s \in \mathcal{SPS}(L(\pi_*))$ in $O(|w|^3)$ time using at most $|w|^2$ membership queries.

By these results, we can conclude that any pattern $\pi \in \mu\Pi$ can be exactly identified in polynomial time using one positive example and membership queries.

Procedure *FindShortest*(w)
Input: a string w in $L(\pi_*)$.
Given: $Mem_{L(\pi_*)}$ for $\pi_* \in \mu\Pi$.
Output: a string s in $\mathcal{SPS}(L(\pi_*))$.
$s := w$;
do
  $s :=$ *Shrink* (s)
while (s is modified)
output s;

Figure 3: Procedure *FindShortest*

Theorem 2. There exists a learning algorithm which exactly identifies every pattern $\pi_* \in \mu\Pi$ in $O(|w|^3)$ time using one positive example $w \in L(\pi_*)$ and at most $|w|^2 + |\pi_*|$ membership queries.

We can extend the procedure *FindShortest* to treat patterns in $k\mu\Pi$ for any fixed $k \geq 1$, and we have the following theorem.

Theorem 3. For a fixed $k \geq 1$, there exists a learning algorithm which exactly identifies every pattern $\pi_* \in k\mu\Pi$ in $O(|w|^{k+2})$ time using one positive example $w \in L(\pi_*)$ and at most $|w|^{k+1} + |\pi_*|^k$ membership queries.

4 Reducing the number of membership queries

In the previous section, we have shown that for any target pattern $\pi_* \in \mu\Pi$, we can find a shortest positive string of $L(\pi_*)$ from one positive example $w \in L(\pi_*)$ with using $|w|^2$ membership queries. In this section, we reduce the number of membership queries into $|w|$ for some subclass of $\mu\Pi$, by showing a sufficient condition that the procedure *Shrink* always outputs a shortest positive string of $L(\pi_*)$.

At first, we introduce the notion of components of a pattern. The *components* of a pattern π are the constant parts of π. The component size of π is the maximum length of components in π. For example, the components of a pattern $\mathtt{aa}\,x_1\,\mathtt{b}\,x_2x_3\mathtt{aba}\,x_4\mathtt{ab}\,x_5$ are aa, b, aba, and ab. Thus the component size of the pattern is 3.

Next we introduce the notion of *critical pattern*, by carefully analyzing the patterns for which the procedure *Shrink* fails to produce a shortest one.

Definition 1. We say that a string $p \in \Sigma^+$ is *critical* if there exists a string $w \in \Sigma^+$ which satisfies the following conditions:

(1) p is a prefix of w,

(2) p does not appear in $w[2:]$,

(3) p appears in $w[2:i-1]w[i+j_{w,p,i}:]$ for some $i \in \{1, 2, \cdots, |p|\}$, where

$$j_{w,p,i} = \max\{j \geq 0 \mid p \text{ appears in } w[:i-1]w[i+k:] \text{ for any } k \in \{0, \cdots, j\}\}.$$

We say that a pattern $\pi \in \mu\Pi$ is *critical* if some component of π is critical.

Example 2. The string $p = \mathtt{abac}$ is critical, since for $w = \mathtt{abacbac}$ and $i = 4$, these conditions hold, where $j_{w,p,i} = 1$. Thus the pattern $\pi_* = x\mathtt{abac}y$ in Example 1 is critical.

Let $\pi = xpy$ be a pattern such that p is not critical, and let w be a string which satisfies the following conditions:

$$w \in L(\pi), \quad \text{(a)}$$
$$w[l:] \notin L(\pi) \text{ for any } l \geq 2. \quad \text{(b)}$$

For $i \in \{2, 3, \cdots, |p|+1\}$, we consider the following string: $w(i) = w[:i-1]w[i+j_i:]$ such that $w[:i-1]w[i+k:] \in L(\pi)$ for any $k \in \{0, \cdots, j_i\}$ and $w[:i-1]w[i+j_i+1:] \notin L(\pi)$. Remark that each string $w(i)$ satisfies (a) and (b) by the definition of critical patterns. The above features of critical strings is used in Lemma 2.

We now show that *Shrink* outputs a shortest positive string of $L(\pi_*)$ from a positive example $w \in L(\pi_*)$ when the target pattern $\pi_* \in \mu\Pi$ is not critical. We denote by s_k the string which is obtained when the oracle $Mem_{L(\pi_*)}$ outputs "*no*" k times in the inner while loop in *Shrink*.

Lemma 2. Let $\pi_* \in \mu\Pi$. If π_* is not critical, there exists a substitution θ such that $\theta(\pi_*[:k]) = s_k[:k]$ for any $k \in \{1, 2, \cdots, |\pi_*|\}$.

Theorem 4. Let $\pi_* \in \mu\Pi$ and $w \in L(\pi_*)$. If π_* is not critical, then $Shrink(w) \in \mathcal{SPS}(L(\pi_*))$.

Corollary 1. Let $\pi_* \in \mu\Pi$ and $w \in L(\pi_*)$. If π_* is not critical, *Shrink* outputs a string in $\mathcal{SPS}(L(\pi_*))$ in $O(|w|^2)$ time from a positive example $w \in L(\pi_*)$ using $|w|$ membership queries.

We can show that the following theorem holds by exhaustively checking each string w with $|w| \leq 3$.

Theorem 5. There is no critical string w with $|w| \leq 3$.

We denoted by $\mu\Pi[l]$ the set of all patterns $\pi \in \mu\Pi$ of which component size are at most l. Then, by Corollary 1 and Theorem 5, the following theorem holds.

Theorem 6. There exists a learning algorithm which exactly identifies every pattern $\pi_* \in \mu\Pi[3]$ in $O(|w|^2)$ time using one positive example $w \in L(\pi_*)$ and exactly $|w| + |\pi_*|$ membership queries.

5 Generating a positive example using membership queries

In this section, we show a nontrivial subclass of $\mu\Pi$ where a learner can effectively enumerate a series of candidates for a positive example. As a consequent, any pattern in the class will be identifiable using membership queries only, *without any initial positive example.* The constraint which enables the effective enumeration consists of two conditions: One is that component size is fixed. The other is that the both ends of the pattern are variable symbols. These conditions are trivially hold for *subsequence languages*, for which we have developed a learning using membership queries only [13].

Definition 2. For an integer $l \geq 0$, let $\mu\Pi[l]^e$ be the set of all patterns $\pi \in \mu\Pi$, such that the component size of π is at most l and both the first and last symbols of π are variable symbols.

The idea to guess a positive example using membership queries is based on the following simple observation.

Remark 1. For an alphabet Σ and a nonnegative integer l, let $p_{\Sigma,l}$ be a string which contains any string in $\Sigma^{[l]}$ as a substring. For instance, the string $\mathtt{aaabbbaabab}$ contains any strings of length at most 3 over $\Sigma = \{\mathtt{a}, \mathtt{b}\}$. Let $p^k_{\Sigma,l}$ be the string $\mathtt{a}p_{\Sigma,l}\mathtt{a}p_{\Sigma,l}\mathtt{a}\ldots\mathtt{a}p_{\Sigma,l}\mathtt{a}$, the k-times repetitions of $p_{\Sigma,l}$ with using $\mathtt{a} \in \Sigma$ as a delimiter symbol. Then for any pattern $\pi \in \mu\Pi[l]^e$, there is an integer $k \leq |\pi|$ such that $p^k_{\Sigma,l} \in L(\pi)$.

Theorem 7. For any fixed $l \geq 0$, any pattern $\pi_* \in \mu\Pi[l]^e$ can be exactly identified in $O(m^3n^3)$ time using at most $m^2n^2 + 2n$ membership queries only, where $m = l(|\Sigma|+1)^l + 2$ and $n = |\pi_*|$.

6 Conclusion

In this paper, we investigated exact identification of a pattern in $k\mu\Pi$ using queries for a fixed $k \geq 1$. We have shown that there exists a learning algorithm which exactly identifies every pattern $\pi_* \in k\mu\Pi$ in $O(|w|^{k+2})$ time using one positive example $w \in L(\pi_*)$ and at most $|w|^{k+1} + |\pi_*|^k$ membership queries.

In Section 4, we introduced the notion of critical patterns. Then we have shown that if a pattern π_* is not critical, a shortest positive string of $L(\pi_*)$ is found from one positive example $w \in L(\pi_*)$ and exactly $|w|$ membership queries. In particular, since any pattern $\pi \in \mu\Pi[3]$ is not critical, we have shown that there exists a learning algorithm which exactly identifies every pattern $\pi_* \in \mu\Pi[3]$ in $O(|w|^2)$ time using one positive example $w \in L(\pi_*)$ and exactly $|w| + |\pi_*|$ membership queries. We summarize these results in Table 1.

		component size	
		at most 3	General
$k\mu\Pi$	$k = 1$	$m + n$ queries $O(m^2)$ time	$m^2 + n$ queries $O(m^3)$ time
	fixed $k \geq 1$	$m^{k+1} + n^k$ queries $O(m^{k+2})$ time	

Table 1: Summary of the results in Section 3 and 4. In the table, m is the length of the positive example and n is the length of a pattern to be identified.

In Section 5, for a fixed $l \geq 0$, we show that any pattern in the class $\mu\Pi[l]^e$ is exactly identifiable using membership queries only.

As future works, we will study a necessary condition of a pattern for which *Shrink* outputs a shortest positive string of $L(\pi_*)$. It may be interesting to study the learnability of a class of unions of pattern languages in our setting.

7 Acknowledgments

The authors wish to acknowledge Takeshi Shinohara, Thomas Zeugmann, Hiroki Ishizaka and Hiroki Arimura for their helpful suggestions and encouragement.

References

[1] D. Angluin. Finding patterns common to a set of strings. *Journal of Computer and System Science*, 21:46–62, 1980.

[2] D. Angluin. Inductive inference of formal languages from positive data. *Information and Control*, 45:117–135, 1980.

[3] D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75:87–106, 1987.

[4] D. Angluin. Queries and concept learning. *Machine Learning*, 2:319–342, 1988.

[5] S. Arikawa, S. Kuhara, S. Miyano, A. Shinohara and T. Shinohara. A learning algorithm for elementary formal systems and its experiments on identification of transmembrane domains. In *Proceedings 25th Hawaii International Conference on System Sciences, Vol. I*, pages 675–684, 1992.

[6] H. Arimura, H. Ishizaka and T. Shinohara. Learning unions of tree patterns using queries. In *Proceedings of 6th Workshop on Algorithmic Learning Theory, Lecture Notes in Artificial Intelligence 997*, pages 66–79, 1995.

[7] A. Bairoch. PROSITE: A dictionary of sites and patterns in proteins. *Nucleic Acids Research*, 19:2241–2245, 1991.

[8] H. Ishizaka, H. Arimura and T. Shinohara. Finding tree patterns consistent with positive and negative examples using queries. In *Proceedings of 5th Workshop on Algorithmic Learning Theory, Lecture Notes in Artificial Intelligence 872*, pages 317–332, 1994.

[9] M. Kearns and L. Pitt. A polynomial-time algorithm for learning k-variable pattern languages from examples. In *Proceedings of the 2nd Annual Conference on Computational Learning Theory*, pages 57–71, 1989.

[10] S. Lange, J. Nessel and R. Wiehagen. Language learning from good examples. In *Proceedings of 5th International Workshop on Algorithmic Learning Theory, Lecture Notes in Artificial Intelligence 872*, pages 423–437, 1994.

[11] S. Lange and R. Wiehagen. Polynomial-time inference of arbitrary pattern languages. *New Generation Computing*, 8(4):361–370, 1991.

[12] A. Marron. Learning pattern languages from a single initial example and from queries. In *Proceedings of the first Annual Conference on Computational Learning Theory*, pages 311–325, 1988.

[13] S. Matsumoto and A. Shinohara. Learning subsequence languages. In *6th European-Japanese Seminar on Information Modelling and Knowledge Bases*, 1996.

[14] S. Miyano, A. Shinohara and T. Shinohara. Which classes of elementary formal systems are polynomial-time learnable? In *Proceedings of 2nd Workshop on Algorithmic Learning Theory*, pages 139–150, 1991.

[15] H. Sakamoto. Language learning from membership queries and characteristic examples. In *Proceedings of 6th International Workshop on Algorithmic Learning Theory, Lecture Notes in Artificial Intelligence 997*, pages 55–65. Springer-Verlag, 1995.

[16] S. Shimozono, A. Shinohara, T. Shinohara, S. Miyano, S. Kuhara and S. Arikawa. Knowledge acquisition from amino acid sequences by machine learning system BONSAI. *Transactions of Information Processing Society of Japan*, 35(10):2009–2018, 1994.

[17] A. Shinohara. Teachability in computational learning. *New Generation Computing*, 8(4):337–347, 1990.

[18] T. Shinohara. Polynomial time inference of extended regular pattern languages. In *RIMS Symposia on Software Science and Engineering(Lecture Notes in Computer Science 147)*, pages 115–127, 1982.

[19] T. Shinohara. Polynomial time inference of pattern languages and its applications. In *Proceedings 7th IBM Symp. Math. Found. Comp. Sci.*, pages 191–209, 1982.

[20] T. Shinohara. Inductive inference from positive data is powerful. In *Proceedings of the 3rd Annual Conference on Computational Learning Theory*, pages 97–110, 1990.

[21] E. Tateishi, O. Maruyama and S. Miyano. Extracting motifs from positive and negative sequence data. In *Proceeding 13th Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Computer Science 1046*, pages 219–230, 1996.

[22] E. Tateishi and S. Miyano. A greedy strategy for finding motifs from positive and negative examples. In *Proceeding First Pacific Symposium on Biocomputing*, pages 599–613. World Scientific Press, 1996.

[23] L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27:1134–1142, 1984.

[24] T. Zeugmann. Average case analysis of pattern language learning algorithm. In *Proceedings of 5th Workshop on Algorithmic Learning Theory, Lecture Notes in Artificial Intelligence 872*, pages 8–9, 1994.

[25] T. Zeugmann. Lange and Wiehagen's pattern language learning algorithm: an average-case analysis with respect to its total learing time. Technical Report RIFIS-TR-CS-111, Aplil 20, Kyushu University, 1995. (to appear in Annals of Mathematics and Artificial Intelligence).